

Relazione di laboratorio di Analisi Numerica: metodi di ricerca zeri

Francesco Genovese, Università di Pavia

8 febbraio 2008

Sommario

Questa relazione di laboratorio di Analisi Numerica (corso dell'A.A. 2007-2008 all'Università di Pavia) presenta alcuni metodi numerici per la ricerca e l'approssimazione degli zeri di funzioni. Sarà presentato il metodo di bisezione e la sua variante *Regula Falsi*. Successivamente si presenterà il metodo di Newton unidimensionale, e in conclusione il metodo delle secanti (che può essere visto come variante del metodo di Newton). Per ciascun metodo sarà esposto l'algoritmo d'implementazione in linguaggio MATLAB, preceduto da brevi cenni teorici.

1 Metodo di bisezione

Il metodo di bisezione è intimamente legato all'importante Teorema degli zeri (o di Bolzano), risultato che sta comunque alla base del problema della ricerca di zeri di funzioni continue in una variabile.

Teorema 1 (degli zeri). *Sia $f : [a, b] \rightarrow \mathbb{R}$ una funzione continua tale che $f(a) < 0 < f(b)$. Allora esiste $c \in (a, b)$ tale che $f(c) = 0$.*

Dimostrazione. L'idea è quella di costruire una successione reale convergente ad un punto che si verifichi essere proprio lo zero della funzione data. Si ponga $a_0 = a$, $b_0 = b$. Poi si definisca $c_0 = (a_0 + b_0)/2$. Se $f(c_0) = 0$ allora non c'è più niente da dimostrare. Se invece $f(c_0) > 0$ si ponga $a_1 = a_0$ e $b_1 = c_0$; al contrario, se $f(c_0) < 0$, si ponga $a_1 = c_0$ e $b_1 = b_0$. Al generico passo k si ponga induttivamente $c_k = (a_k + b_k)/2$. Se $f(c_k) = 0$ non c'è più nulla da dimostrare, se $f(c_k) > 0$ si ponga $a_{k+1} = a_k$ e $b_{k+1} = c_k$, se invece $f(c_k) < 0$ si ponga $a_{k+1} = c_k$ e $b_{k+1} = b_k$. Risultano così costruite induttivamente le tre successioni $\{a_n\}$, $\{b_n\}$ e $\{c_n\}$. Si vede immediatamente che $\{a_n\}$ è nondecreciente, $\{b_n\}$ è noncrescente, e nondimeno $a_0 \leq a_n \leq c_n \leq b_n \leq b_0$ per ogni n (quindi per il teorema delle successioni monotone $\lim_{n \rightarrow +\infty} a_n$ e $\lim_{n \rightarrow +\infty} b_n$ esistono finiti). Si nota poi che $b_n - a_n = (b_{n-1} - a_{n-1})/2$, e

di conseguenza $b_n - a_n = (b_0 - a_0)/2^n$. Quindi $\lim_{n \rightarrow +\infty} (b_n - a_n) = 0 = \lim_{n \rightarrow +\infty} b_n - \lim_{n \rightarrow +\infty} a_n$, cioè $\lim_{n \rightarrow +\infty} a_n = \lim_{n \rightarrow +\infty} b_n$. Possiamo allora applicare il teorema dei carabinieri e concludere che

$$\lim_{n \rightarrow +\infty} a_n = \lim_{n \rightarrow +\infty} b_n = \lim_{n \rightarrow +\infty} c_n$$

Sia allora c tale limite comune. La continuità della funzione f ci assicura che $f(c) = \lim_{n \rightarrow +\infty} f(a_n) = \lim_{n \rightarrow +\infty} f(b_n)$. Nondimeno il fatto che $[a, b]$ sia chiuso assicura che $c \in [a, b]$. D'altra parte, per costruzione induttiva si ha che $f(a_n) < 0 < f(b_n)$. Quindi possiamo applicare il teorema di conservazione delle disuguaglianze e affermare

$$f(c) = \lim_{n \rightarrow +\infty} f(a_n) \leq 0 \leq \lim_{n \rightarrow +\infty} f(b_n) = f(c)$$

Quindi $f(c) \leq 0 \leq f(c)$, di conseguenza $f(c) = 0$. Siccome poi a e b non sono zeri di f , deve essere che $c \in (a, b)$, come volevamo. Ovviamente il teorema vale anche nell'ipotesi che $f(a) > 0 > f(b)$, basta applicare il procedimento visto a $-f$, sicuri del fatto che gli zeri di f sono tutti e soli quelli di $-f$. \square

Ho ritenuto opportuno scrivere tutta la dimostrazione del Teorema degli zeri proprio perché, come pure accennavo, il metodo di bisezione non consiste in altro che nella costruzione delle successioni $\{a_n\}$, $\{b_n\}$ e $\{c_n\}$ come nel procedimento utilizzato nella dimostrazione stessa. È chiaro che il metodo ha il vantaggio di convergere sempre ad uno zero della funzione assegnata.

Per studiare la rapidità di convergenza del metodo, supponiamo che l'errore compiuto al passo n -esimo sia stimato con l'ampiezza dell'intervallo $[a_n, b_n]$. Di conseguenza:

$$e_n := b_n - a_n = \frac{b - a}{2^n}$$

Come si è anche visto nella dimostrazione, $\{e_n\}$ converge a 0, come ci aspetteremmo. Dunque si ricava immediatamente che:

$$\left| \frac{e_{n+1}}{e_n} \right| = \frac{1}{2}$$

In sostanza, il metodo di bisezione ha lo svantaggio di essere solo del primo ordine, la convergenza è sì assicurata ma è piuttosto lenta. Un'altra peculiarità interessante del metodo è che, fissata una tolleranza $\varepsilon > 0$, è possibile conoscere a priori quante iterazioni sono necessarie per raggiungerla. Infatti, fissato ε , per la convergenza della successione $\{e_n\}$, possiamo trovare m tale che si abbia $e_m \leq \varepsilon$. Di conseguenza, con un rapido calcolo si trova che:

$$e_m = \frac{b - a}{2^m} \leq \varepsilon \Leftrightarrow m \geq \log_2 \frac{b - a}{\varepsilon}$$

Quindi il metodo può essere fermato al passo $m^* = \lceil \log_2 \frac{b-a}{\varepsilon} \rceil$ per ottenere la tolleranza ε sull'errore compiuto dal metodo. Tale fatto può essere usato opportunamente come test d'arresto nell'algoritmo.

Algoritmo 1 (metodo di bisezione). *Il seguente algoritmo implementa in MATLAB il metodo di bisezione. È costruito in modo che tutti i termini delle successioni $\{a_n\}$, $\{b_n\}$ e $\{c_n\}$ siano immagazzinati in opportuni vettori e costituisce quanto di più vicino al procedimento dimostrativo del Teorema degli zeri.*

```
function [zero, i]=bisez(f, a, b, toll1, toll2)
    g=inline(f);
    i=1;
    nmax=ceil(log2((b-a)/toll1))+1 %test di arresto 1
    if (g(a)*g(b) > 0)
        display('Impostare un diverso intervallo')
        break
    end
    if (g(a)==0)
        zero=g(a)
        break
    end
    if (g(b)==0)
        zero=g(b)
        break
    end
    if (g(a) > 0)
        display('mettere -f in input')
        break
    end
    if (g(a)*g(b) < 0)
        a(1) = a;
        b(1) = b;
        for (i=1:nmax)
            c(i) = ( a(i) + b(i) )/2;
            if (g(c(i))==0)
                zero=c(i)
                break
            end
            if (g(c(i)) > 0)
                a(i+1)=a(i);
                b(i+1)=c(i);
            end
            if (g(c(i)) < 0)
                a(i+1)=c(i);
            end
        end
    end
end
```

```

        b(i+1)=b(i);
    end
    if (i > 1)
        if (abs(g(c(i))-g(c(i-1))) < toll2) %test di arresto 2
            zero=c(i);
            display(zero)
            break
        end
    end
end
end
end
display(i)
zero=c(i);
display(zero)

```

Notiamo che il “test di arresto 1” è quello caratteristico del metodo di bisezione, mentre il “test di arresto 2” è un test classico utilizzato nei metodi iterativi di qualunque sorta.

1.1 Metodo *Regula Falsi*

Si tratta di una modifica al metodo di bisezione che ha il vantaggio di valutare non solo il segno, ma anche i valori della funzione di cui si vuole approssimare lo zero. Le ipotesi sono identiche a quelle del metodo di bisezione, cioè: $f : [a, b] \rightarrow \mathbb{R}$ continua e tale che $f(a) < 0 < f(b)$. Si procede formalmente come per il metodo di bisezione, eccetto che nella definizione della successione $\{c_n\}$. Al passo k , invece di scegliere c_k come il punto medio di $[a_k, b_k]$, si sceglie c_k come lo zero della retta passante per $(a_k, f(a_k))$ e $(b_k, f(b_k))$. Tale retta si verifica rapidamente avere espressione analitica:

$$y = f(a_k) + \frac{f(b_k) - f(a_k)}{b_k - a_k}(x - a_k)$$

E il suo zero si ricava essere:

$$c_k = \frac{a_k f(b_k) - b_k f(a_k)}{f(b_k) - f(a_k)}$$

Si può notare che il metodo *Regula Falsi* è effettivamente una generalizzazione del metodo di bisezione, giacché il punto medio di $[a_k, b_k]$ può essere visto come zero della retta passante per $(a_k, \text{sign } f(a_k))$ e $(b_k, \text{sign } f(b_k))$, ove ricordiamo che $\text{sign } f(a_k) = -1$ e $\text{sign } f(b_k) = 1$ per ogni k , per costruzione.

Il problema del metodo *Regula Falsi* è che in generale non è vero che $(b_n - a_n) \rightarrow 0$ per $n \rightarrow +\infty$. A tale inconveniente si può ovviare modificando leggermente il metodo. L’idea è che, se $f(c_i)$ (generico passo $i > 1$) ha lo stesso segno di $f(c_{i-1})$, si calcoli c_{i+1} come zero della retta passante per:

$(a_{i+1}, f(a_{i+1}/2))$ e $(b_{i+1}, f(b_{i+1}))$ se tale segno è positivo, $(a_{i+1}, f(a_{i+1}))$ e $(b_{i+1}, f(b_{i+1})/2)$ se tale segno è negativo.

Algoritmo 2 (metodo *Regula Falsi* modificato). *Ecco dunque l'algoritmo del metodo, con implementata la modifica di cui sopra.*

```
function [zero, i]=regfal(f, a, b, nmax, toll)
    g=inline(f);
    i=1;
    if (g(a)*g(b) > 0)
        ('Impostare un diverso intervallo')
        break
    end
    if (g(a)==0)
        zero=g(a)
        break
    end
    if (g(b) == 0)
        zero=g(b)
        break
    end
    if (g(a) > 0)
        ('mettere -f in input')
        break
    end
    if (g(a)*g(b) < 0)
        a(1) = a;    %primo passo iterazione scritto per esteso
        b(1) = b;
        c(1) = (a(1)*g(b(1)) - b(1)*g(a(1)))/(g(b(1))-g(a(1)));
        if (g(c(1))==0)
            zero=c(1)
            break
        end
        if (g(c(1)) > 0)
            a(2)=a(1);
            b(2)=c(1);
        end
        if (g(c(1)) < 0)
            a(2)=c(1);
            b(2)=b(1);
        end
        end
        L= g(a(2));
        R = g(b(2));
        for (i=2:nmax)
            c(i) = (a(i)*R - b(i)*L)/(R-L);
```

```

    if (g(c(i))==0)
        zero=c(i)
        break
    end
    if (g(c(i)) > 0)
        a(i+1)=a(i);
        b(i+1)=c(i);
        R=g(b(i+1));
        L=g(a(i+1));
        if g(c(i-1))*g(c(i)) > 0
            L=L/2;    %ecco la modifica
        end
    end
    if (g(c(i)) < 0)
        a(i+1)=c(i);
        b(i+1)=b(i);
        R=g(b(i+1));
        L=g(a(i+1));
        if g(c(i-1))*g(c(i)) > 0
            R=R/2;    %ecco la modifica
        end
    end
    if (i > 1)
        if (abs(g(c(i))-g(c(i-1))) < toll) %test arresto
            zero=c(i)
            break
        end
    end
end
end
display(i)
zero=c(i)
display(zero)

```

2 Metodo di Newton

Il metodo di Newton permette di approssimare lo zero di una funzione data con una rapidità di convergenza maggiore rispetto ai metodi visti sinora, questo utilizzando la derivata prima della funzione in gioco, ovviamente se le ipotesi lo permettono. Sia dunque $f : [a, b] \rightarrow \mathbb{R}$, $f \in C^1(a, b)$, ed esista $\alpha \in (a, b)$ tale che $f(\alpha) = 0$. L'idea geometrica del metodo è molto semplice: si fissa un punto iniziale $x_0 \in (a, b)$ ad arbitrio. Al generico passo k si approssima la funzione data f con la retta tangente a f nel punto

$(x_k, f(x_k))$). Dall'Analisi è ben noto che tale retta ha equazione:

$$y = f(x_k) + f'(x_k)(x - x_k)$$

A questo punto, si definisce l'elemento x_{k+1} della successione approssimante come lo zero di tale retta tangente. L'iterazione del metodo di Newton risulta dunque essere, dopo un semplice calcolo:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Si può dimostrare che, se $f \in C^2(a, b)$, se α è uno zero semplice della funzione f e il metodo converge, allora è del secondo ordine, in particolare:

$$\lim_{k \rightarrow +\infty} \frac{e_{k+1}}{e_k^2} = \frac{1}{2} \frac{f''(\alpha)}{f'(\alpha)}$$

ove $e_k := x_k - \alpha$ è l'errore al passo k -esimo. Nondimeno, si può mostrare che il metodo di Newton converge se il punto di partenza x_0 è scelto "sufficientemente vicino" allo zero α ; per questo motivo, è comune utilizzare il metodo di bisezione, utile a restringere opportunamente l'intervallo in cui cercare lo zero della funzione, prima di applicare il metodo di Newton, che ha convergenza più rapida.

Algoritmo 3 (metodo di Newton). *Di seguito, l'implementazione MATLAB per l'algoritmo di Newton. Bisogna notare che MATLAB non è in grado di calcolare le derivate, quindi nell'input l'utente è costretto a inserirla a mano.*

```
function [zero, fz, iter, xk, fk]= newt(f,fd,x0,toll,niter)
g = inline(f);
gd=inline(fd);
iter=1; %passo iniziale
xk(iter)=x0; %fai esplicitamente la prima iterazione
y=feval(g,x0);
dy=feval(gd,x0);
fk(iter)=feval(g, x0); %g valutata in x0
for iter=2:niter
    xk(iter)= xk(iter-1) - y/dy;
    fk(iter)=feval(g,xk(iter));
    if abs(xk(iter)-xk(iter-1)) < toll %test di arresto
        break
    end
    y=feval(g,xk(iter)); %preparazione al passo successivo
    dy=feval(gd,xk(iter));
end
zero=xk(iter);
```

```

fz=feval(g,zero); %valore della f in zero
display(zero)
display(fz)
display(xk)

```

2.1 Metodo delle secanti

Questa variante del metodo di Newton consiste nell'approssimare la funzione con opportune rette secanti, costruite iterativamente. Sia dunque $f : [a, b] \rightarrow \mathbb{R}$ regolare quanto serve, ed esista $\alpha \in (a, b)$ tale che $f(\alpha)=0$. Si scelgano arbitrariamente $x_0, x_1 \in (a, b)$. Al generico passo k , si consideri la retta passante per $(x_{k-1}, f(x_{k-1}))$ e $(x_k, f(x_k))$. Essa ha espressione analitica:

$$y = f(x_{k-1}) + \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_{k-1})$$

Si stabilisce x_{k+1} lo zero di quella retta, che risulta quindi essere, con un semplice calcolo:

$$x_{k+1} = x_k - f(x_k) \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

La formula è valida se ovviamente $f(x_k) \neq f(x_{k-1})$.

Algoritmo 4 (metodo delle secanti). *Di seguito, l'implementazione in MATLAB del metodo delle secanti.*

```

function [zero, i]=secant(f,x0,x1,nmax,toll)
g=inline(f);
i=2;
x(1)=x0;
x(2)=x1;
for i=2:nmax
    if (g(x(i)) == g(x(i-1)))
        break
    else
        x(i+1) = x(i)-g(x(i))*((x(i)-x(i-1))/(g(x(i))-g(x(i-1))));
    end
    if (abs(x(i+1)-x(i)) < toll)
        zero=x(i+1)
        break
    end
end
end
zero=x(i+1);
display(i)

```